



Functions

Chris Gregg

Based on slides by Chris Piech and Mehran Sahami
CS106A, Stanford University

Learn How To:

1. Write a function that takes in input
2. Write a function that gives back output
3. Trace function calls using stacks



Calling functions

```
turn_right()
```

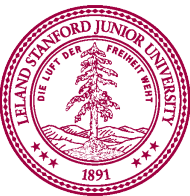
```
move()
```

```
input("string please! ")
```

```
print("hello world")
```

```
float("0.42")
```

```
math.sqrt(25)
```

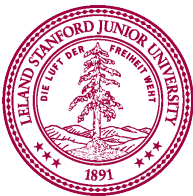


Defining a function

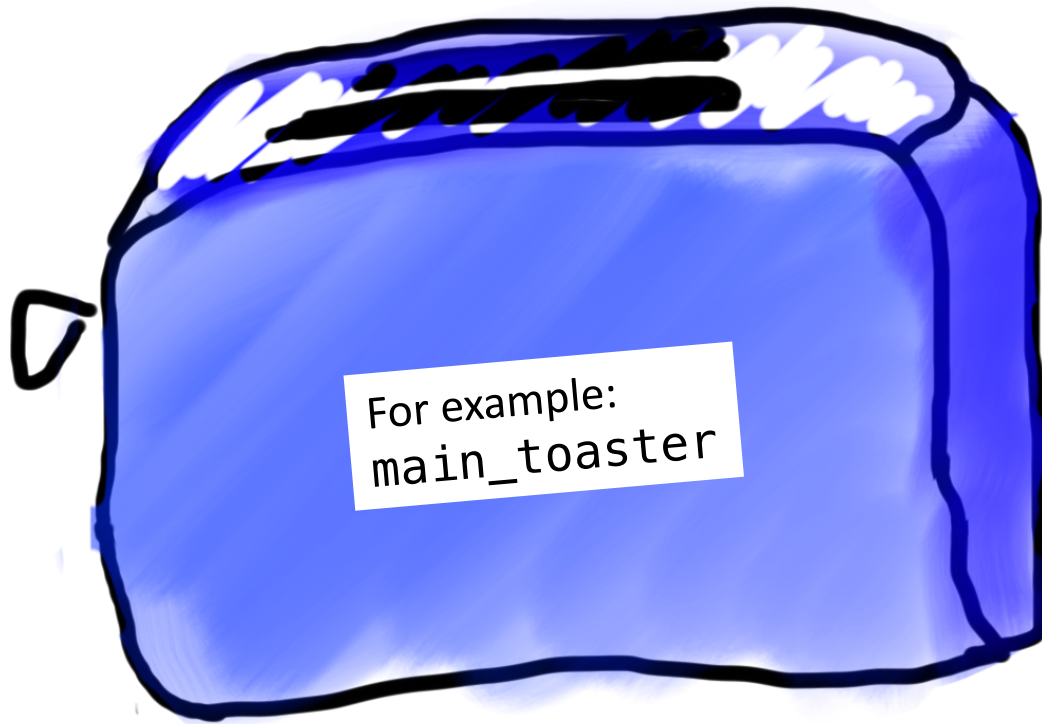
```
def turn_right():  
    turn_left()  
    turn_left()  
    turn_left()
```



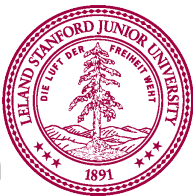
Big difference with python functions:
Python functions can **take in data**, and can **return data**!



Toasters are functions



- Thanks Mehran



Toasters are functions



parameter



Toasters are functions



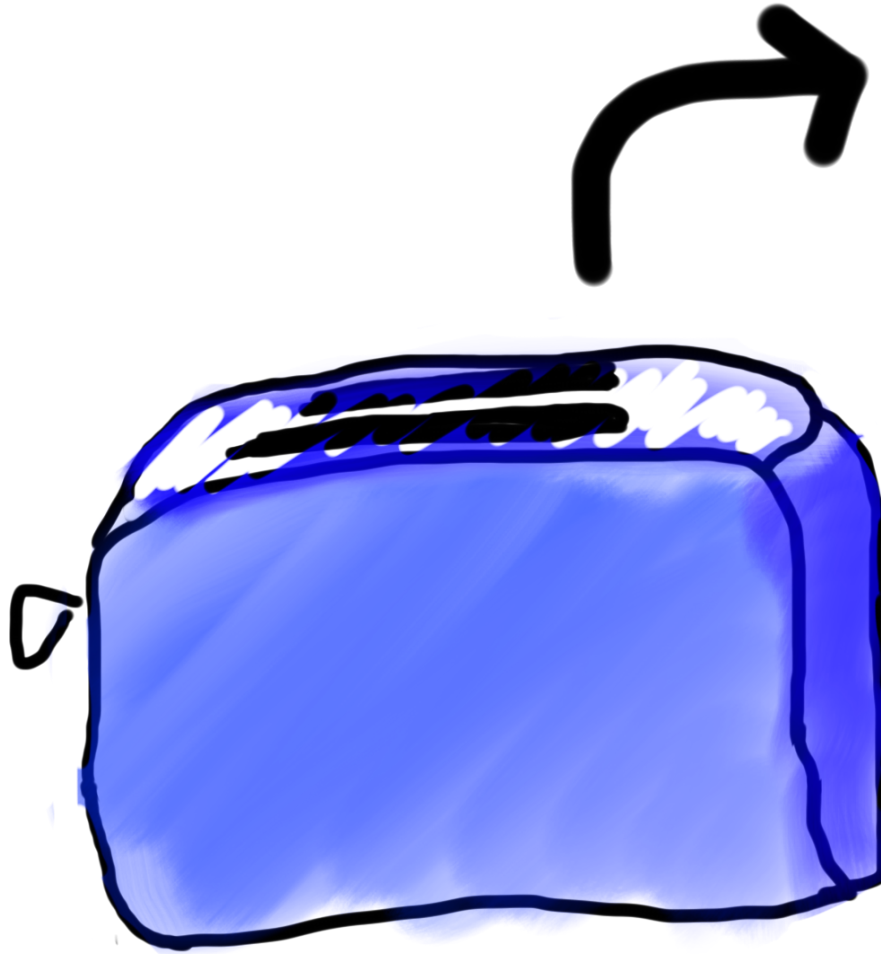
parameter



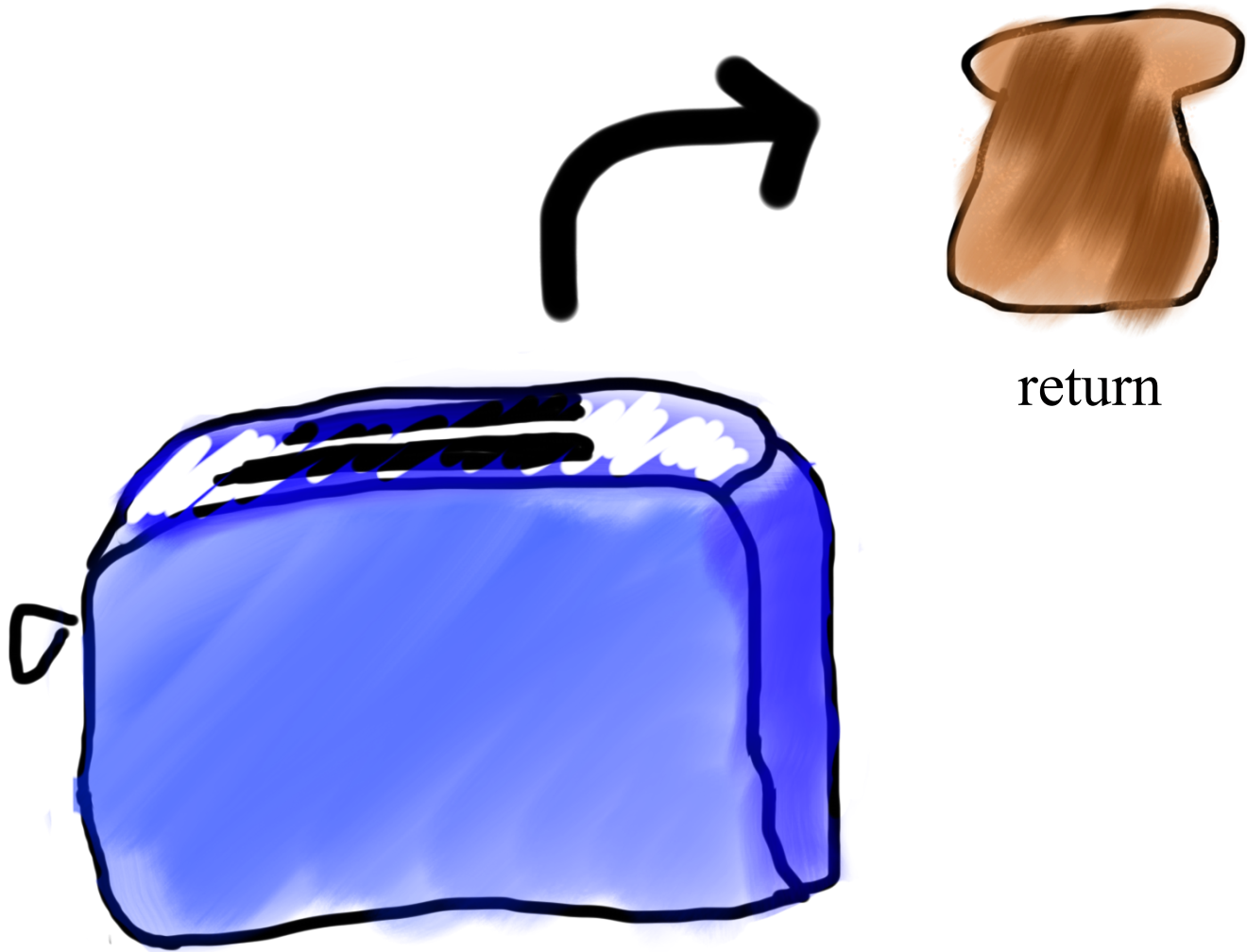
Toasters are functions



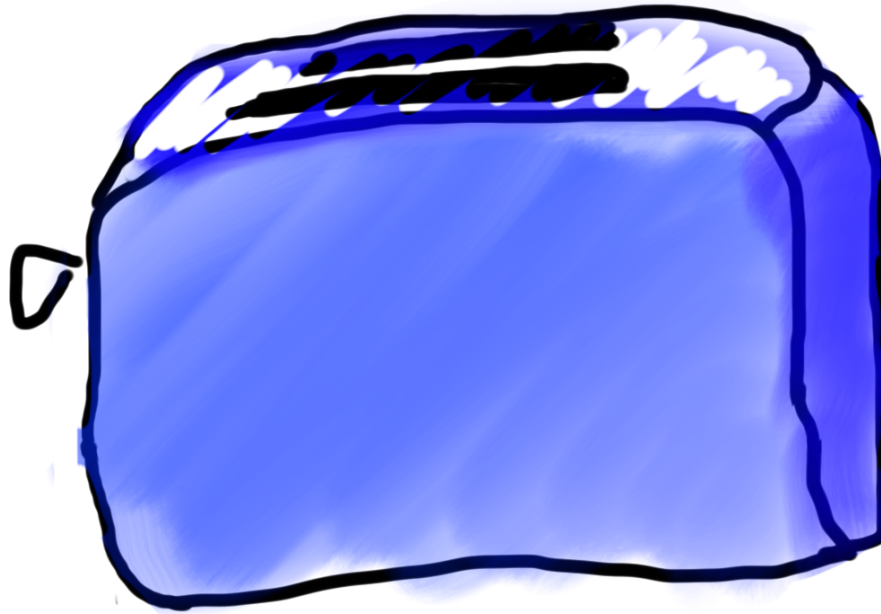
Toasters are functions



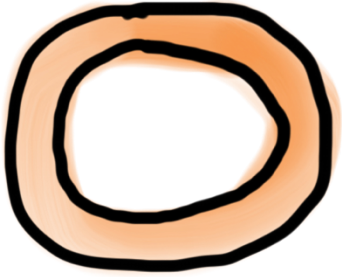
Toasters are functions



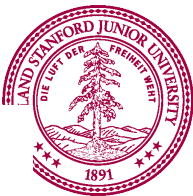
Toasters are functions



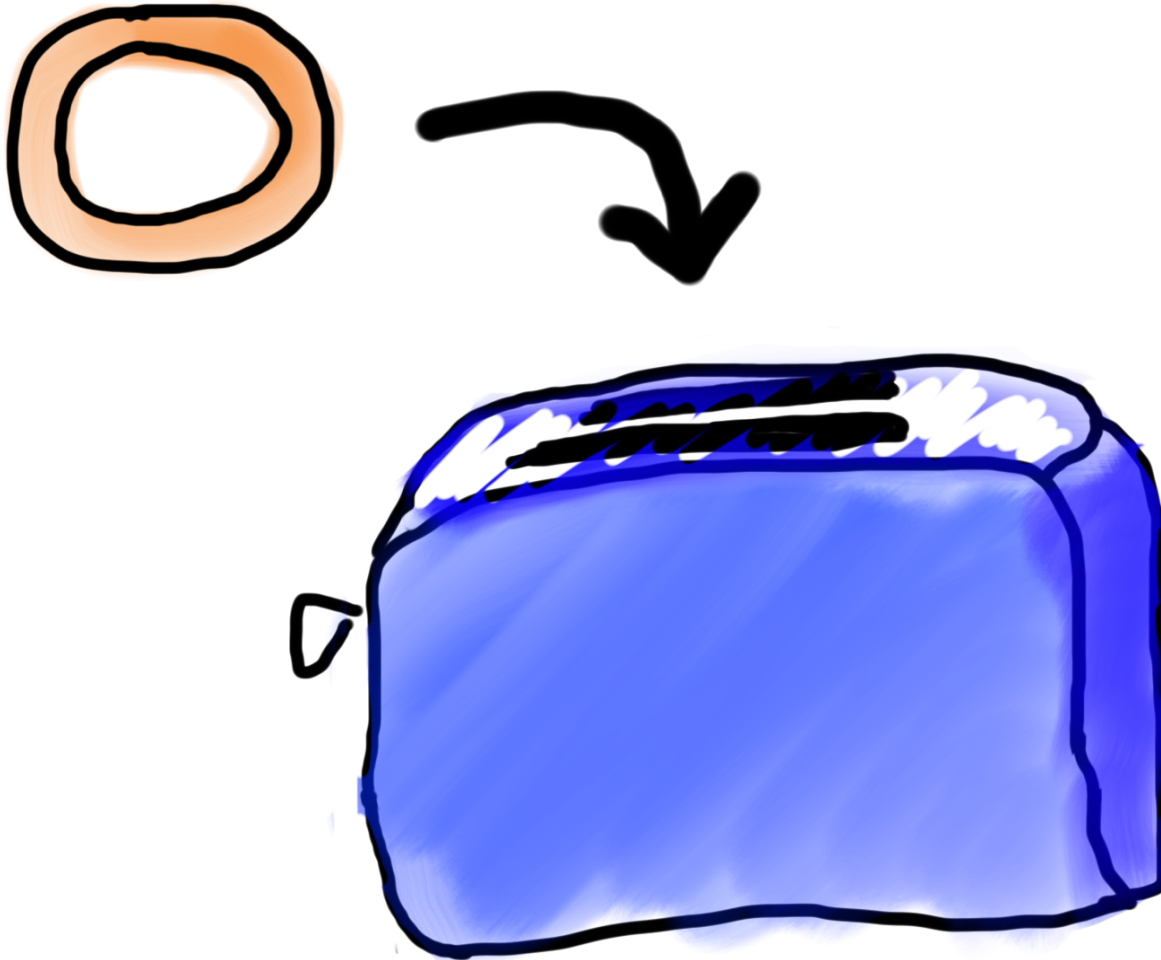
Toasters are functions



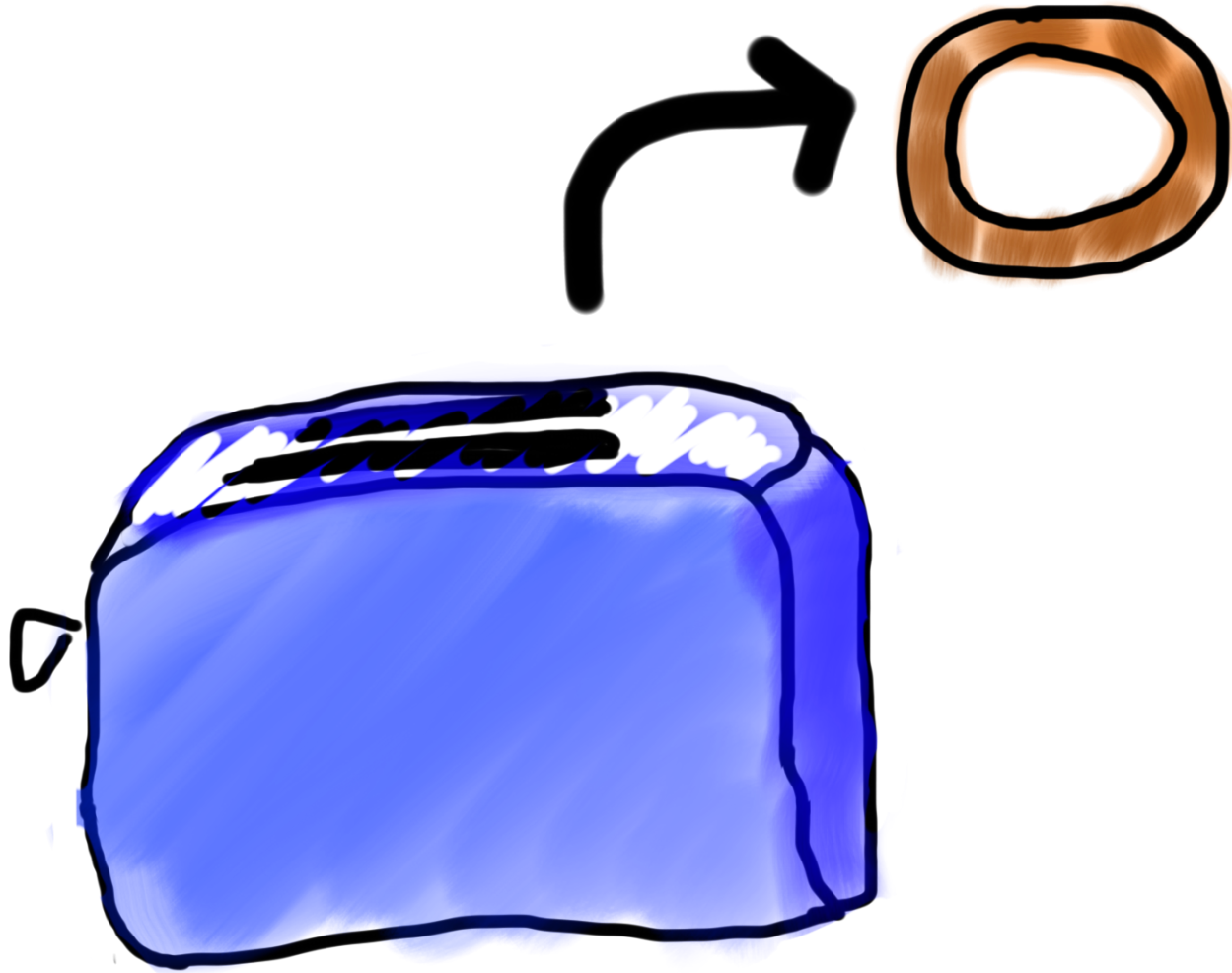
* You don't need a second toaster if you want to toast bagels. Use the same one.



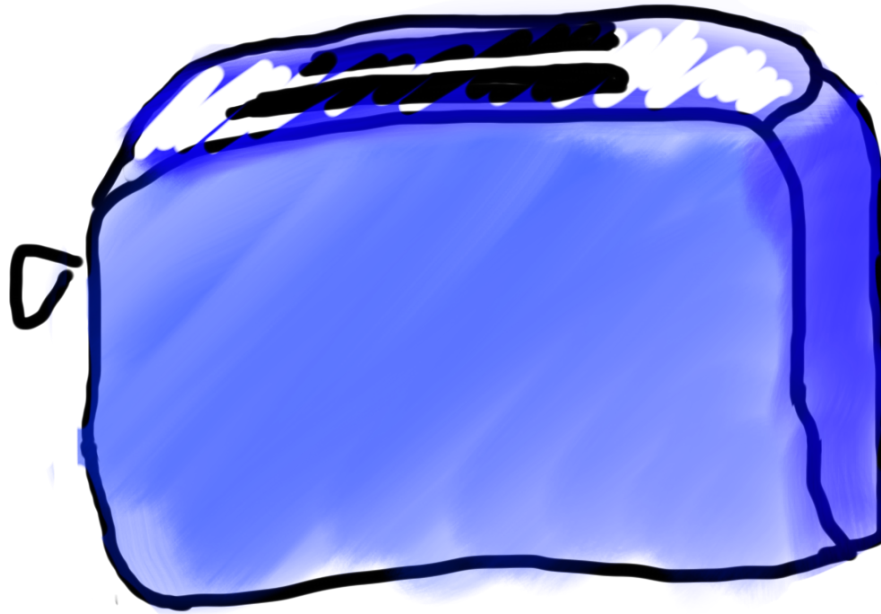
Toasters are functions



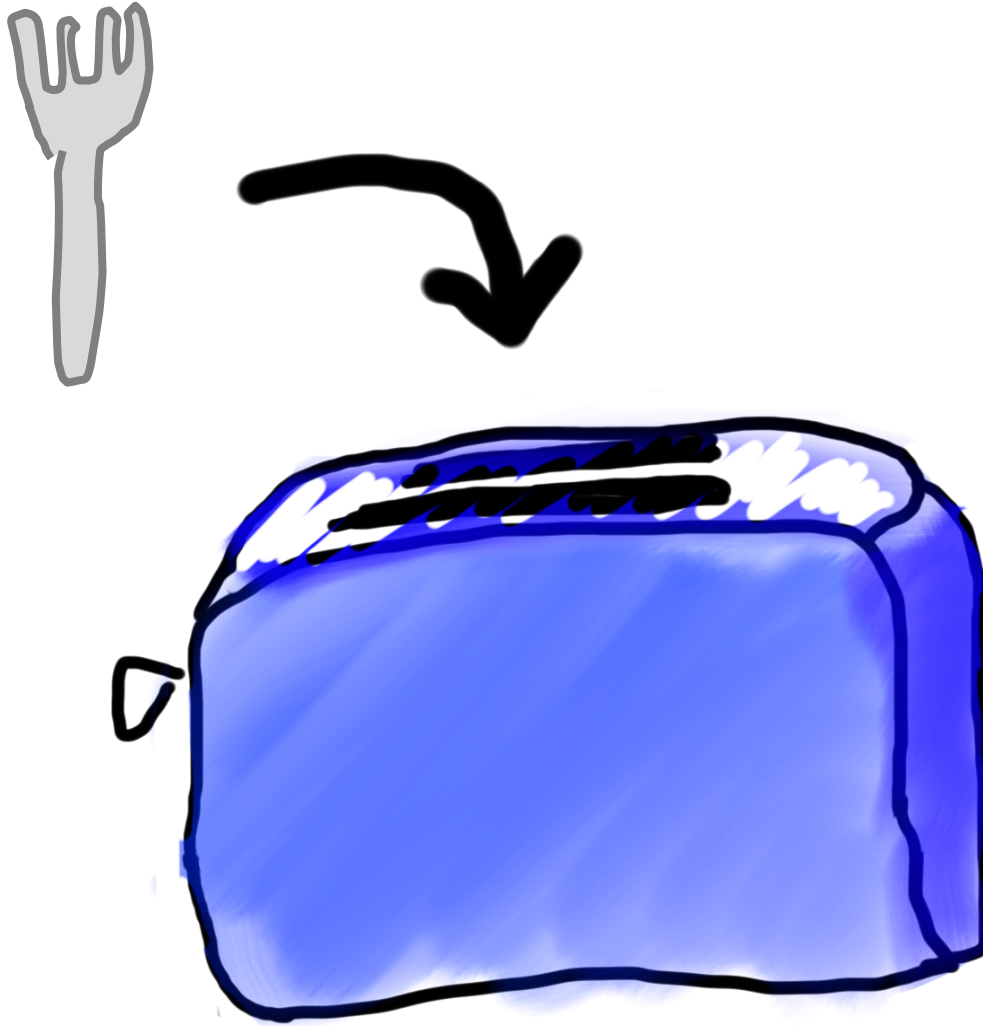
Toasters are functions



Toasters are functions



Toasters are functions



Toasters are functions



functions are Like Toasters



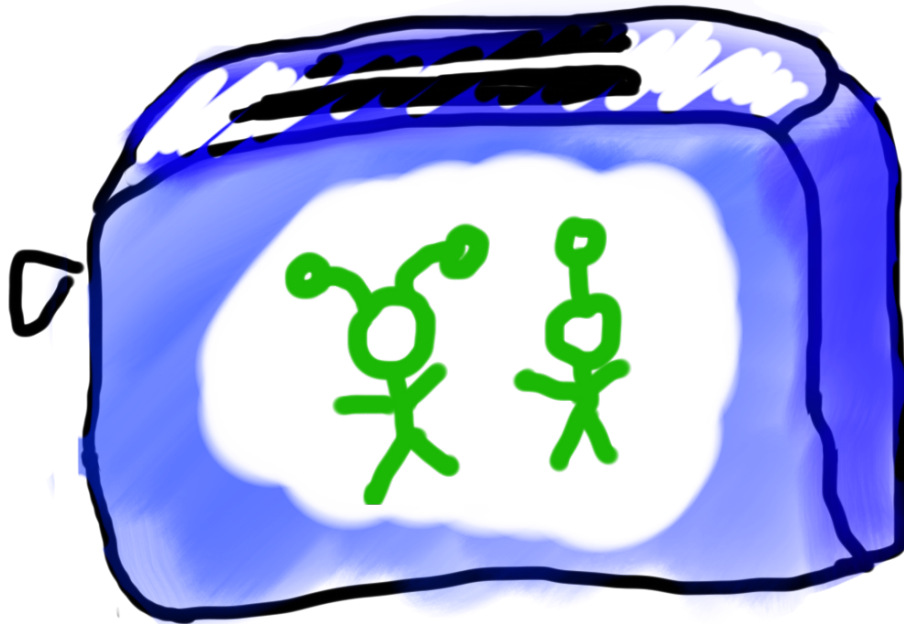
functions are Like Toasters



functions are Like Toasters



functions are Like Toasters



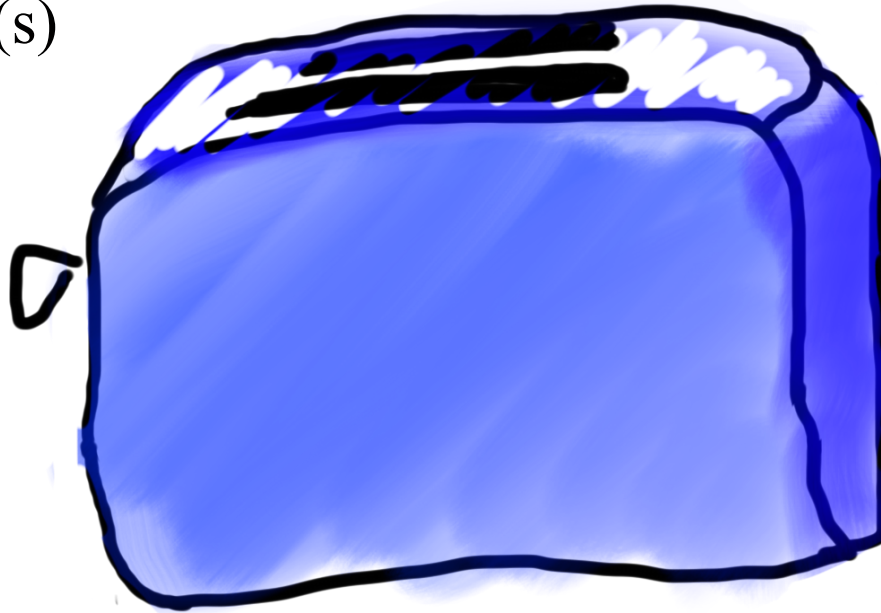
functions are Like Toasters



parameter(s)



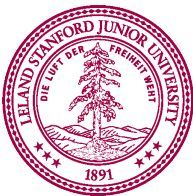
return



Formally

```
def name_of_function (parameters) :  
    statements  
    # optionally  
    return value
```

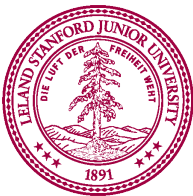
- *name*: what you call the function
- *parameters*: information passed into function
- *return*: information given back from the function



Classic Challenge for CS106A



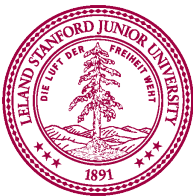
Perhaps the
most
underrated
concept by
students



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

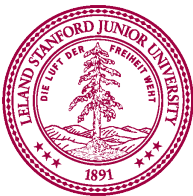


Anatomy of a function

```
def main():    function "call"  
    mid = average(5.0, 10.2)  
    print(mid)
```

function "definition"

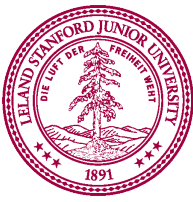
```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def nameaverage(a, b):  
    sum = a + b  
    return sum / 2
```



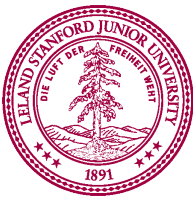
Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

Input given

Input expected

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



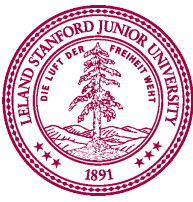
Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

Arguments

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Parameters



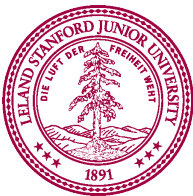
Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):
```

```
    sum = a + b  
    return sum / 2
```

body

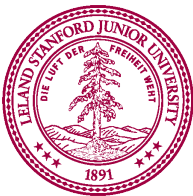


Anatomy of a function

```
def main():    This call “evaluates” to the value returned  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

Ends the function and gives
back a value



Anatomy of a function

Also a function call

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

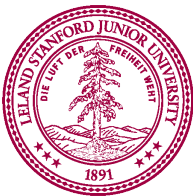


Anatomy of a function

No parameters (expects no input)

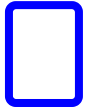
```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



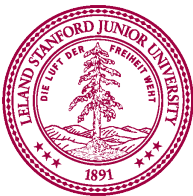
Anatomy of a function

```
def main():  
    mid = average(5.0, 10.2)  
    print(mid)
```



When a function ends it “returns”

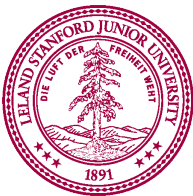
```
def average(a, b):  
    sum = a + b  
    return sum / 2
```



Parameters



Parameters let
you provide a
function some
information
when you are
calling it.



Is returning
the same as printing?

Is returning
the same as printing?

NO

Learn by Example



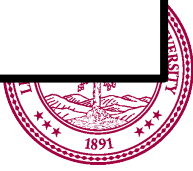
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



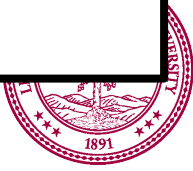
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



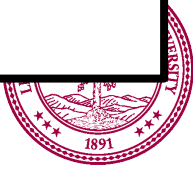
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



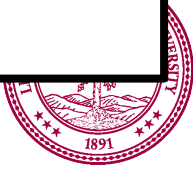
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py
```



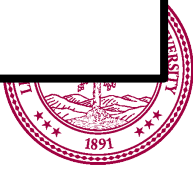
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class
```



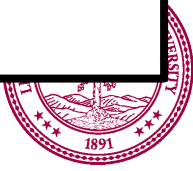
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



No Parameter, No Return

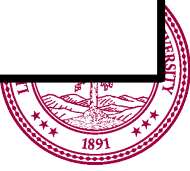
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```



```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



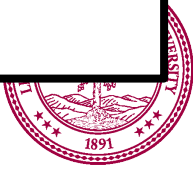
No Parameter, No Return

```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```

terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```



No Parameter, No Return

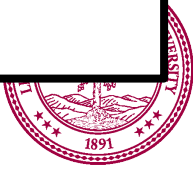
```
def print_intro():  
    print("Welcome to class")  
    print("It's the best part of my day.")
```

```
def main():  
    print_intro()
```



terminal

```
> python intro.py  
Welcome to class  
It's the best part of my day
```

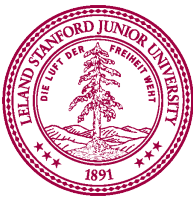


Parameter Example

```
def print_opinion(num) :  
    if (num == 5) :  
        print("I love 5!")  
    else:  
        print("Whatever")  
  
def main() :  
    print_opinion(5)
```

terminal

```
> python opinion.py
```



Parameter Example

main memory

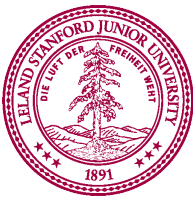
No variables

```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py
```



Parameter Example

main memory

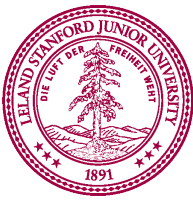
No variables

```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

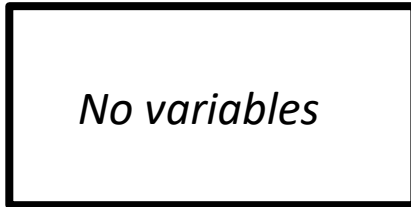
terminal

```
> python opinion.py
```



Parameter Example

main memory



print_opinion memory

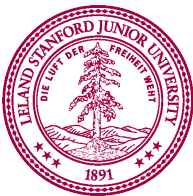


terminal

```
> python opinion.py
```

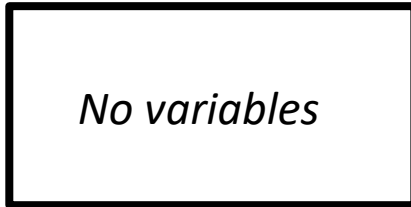
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

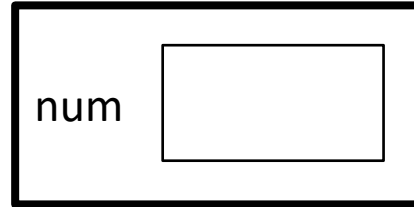


Parameter Example

main memory



print_opinion memory

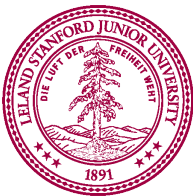


terminal

> python opinion.py

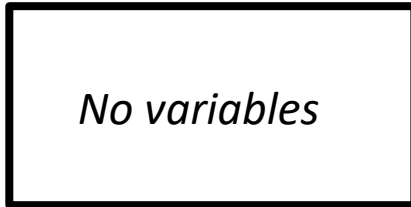
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

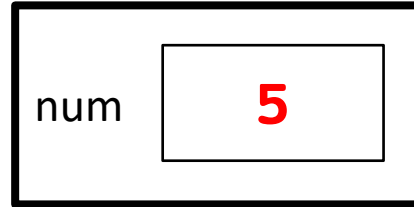


Parameter Example

main memory



print_opinion memory

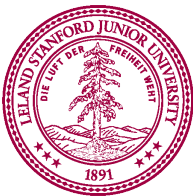


terminal

> python opinion.py

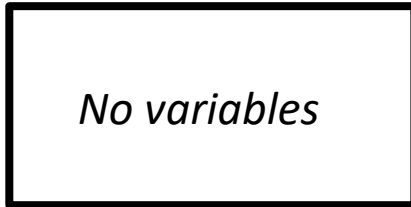
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

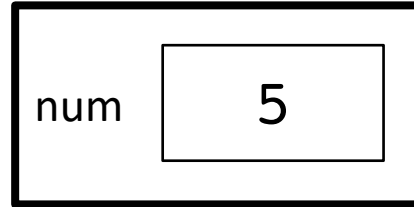


Parameter Example

main memory



print_opinion memory

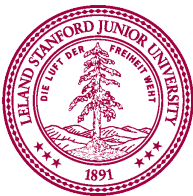


terminal

> python opinion.py

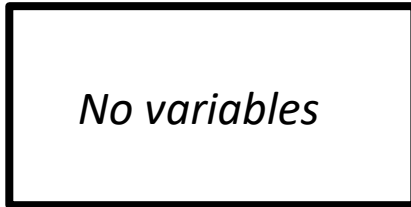
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

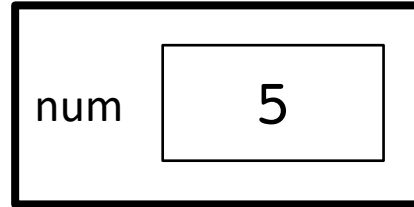


Parameter Example

main memory



print_opinion memory

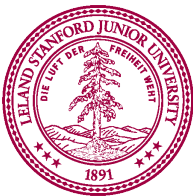


terminal

```
> python opinion.py  
I love 5!
```

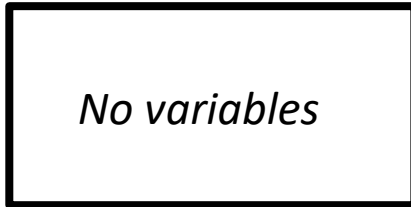
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

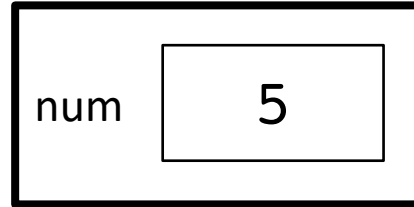


Parameter Example

main memory



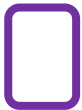
print_opinion memory



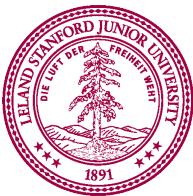
terminal

```
> python opinion.py  
I love 5!
```

```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```



```
def main():  
    print_opinion(5)
```

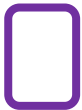


Parameter Example

main memory

No variables

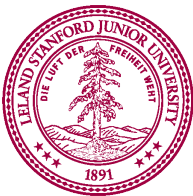
```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```



```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py  
I love 5!
```



Parameter Example

main memory

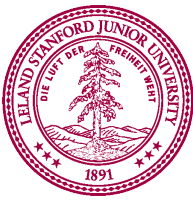
No variables

```
def print_opinion(num):  
    if(num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```

terminal

```
> python opinion.py  
I love 5!
```



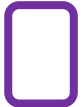
Parameter Example

main memory

No variables

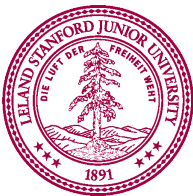
```
def print_opinion(num):  
    if (num == 5):  
        print("I love 5!")  
    else:  
        print("Whatever")
```

```
def main():  
    print_opinion(5)
```



terminal

```
> python opinion.py  
I love 5!
```

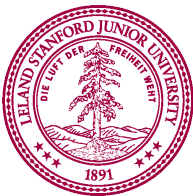


Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters  
  
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory

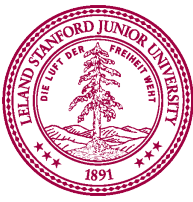
No variables

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory

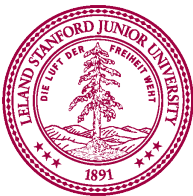
No variables

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

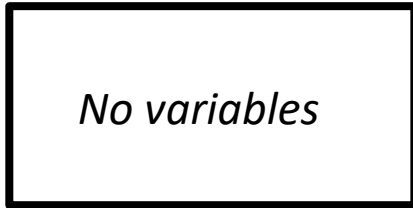
terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory



metersToCm memory

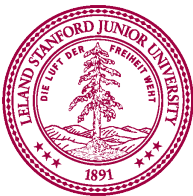


```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

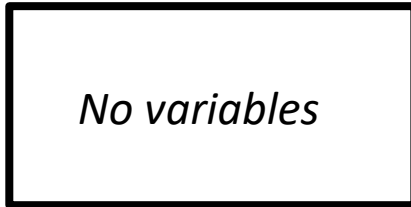
terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory



metresToCm memory

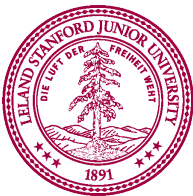


terminal

```
> python3 m2cm.py
```

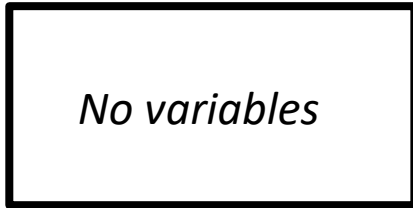
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

main memory



metersToCm memory



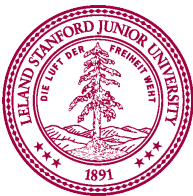
terminal

```
> python3 m2cm.py
```

```
def meters_to_cm(meters):  
    return 100 * meters
```

520.0

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```



Parameter and Return Example

main memory

No variables

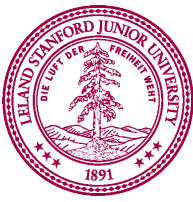
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

520.0

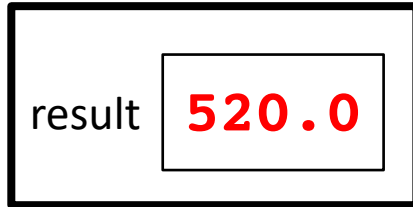
terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory

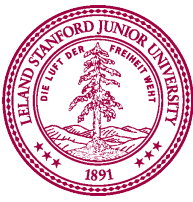


```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

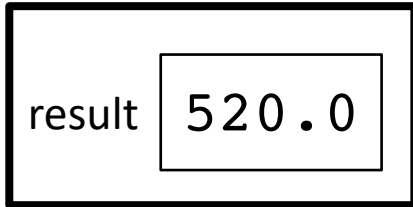
terminal

```
> python3 m2cm.py
```



Parameter and Return Example

main memory

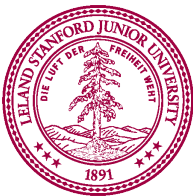


```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    result = meters_to_cm(5.2)  
    print(result)
```

terminal

```
> python3 m2cm.py  
520.0
```

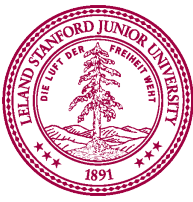


Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters  
  
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python3 m2cm.py
```



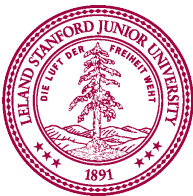
Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python3 m2cm.py
```



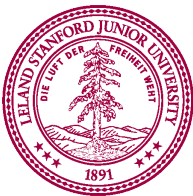
Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python3 m2cm.py
```



Parameter and Return Example

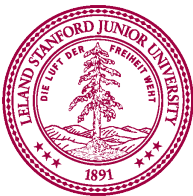
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

520.0

terminal

```
> python3 m2cm.py
```



Parameter and Return Example

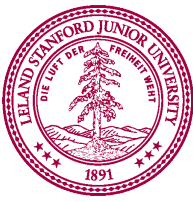
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

520.0

terminal

```
> python3 m2cm.py  
520.0
```



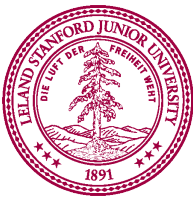
Parameter and Return Example

```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

terminal

```
> python3 m2cm.py  
520.0
```



Parameter and Return Example

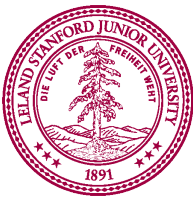
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

910.0

terminal

```
> python3 m2cm.py  
520.0
```



Parameter and Return Example

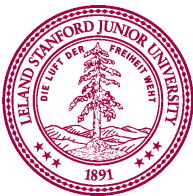
```
def meters_to_cm(meters):  
    return 100 * meters
```

```
def main():  
    print(meters_to_cm(5.2))  
    print(meters_to_cm(9.1))
```

910.0

terminal

```
> python3 m2cm.py  
520.0  
910.0
```



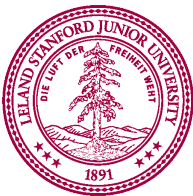
Contrasting Case:

How is this function

```
def meters_to_cm_case1(meters):  
    return 100 * meters
```

Different than this function?

```
def meters_to_cm_case2(meters):  
    print(100 * meters)
```



Is returning
the same as printing?

Is returning
the same as printing?

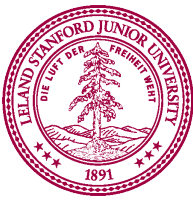
NO

Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

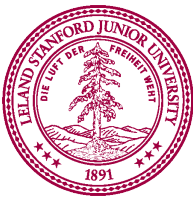
No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

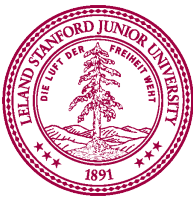
No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

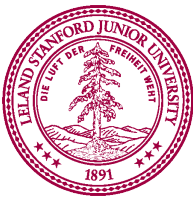
No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

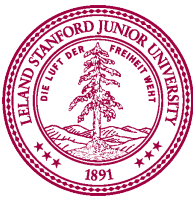
main memory

No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(5, 1)
```

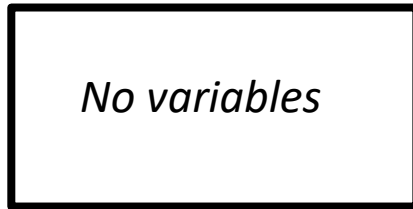
terminal

```
> python maxmax.py
```

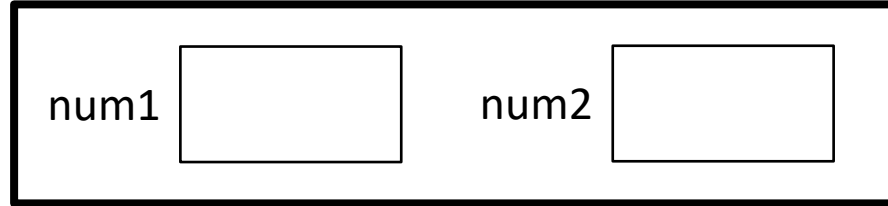


Multiple Return Statements

main memory



max memory

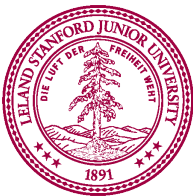


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

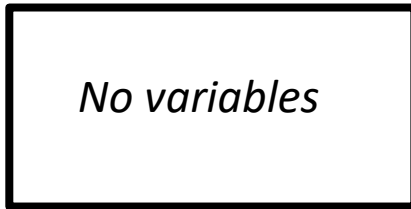
terminal

```
> python maxmax.py
```

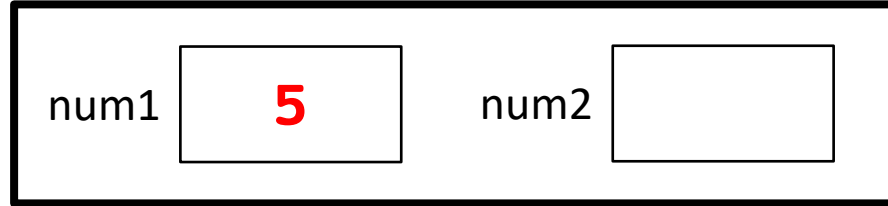


Multiple Return Statements

main memory



max memory

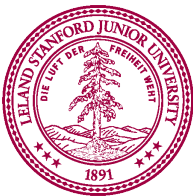


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

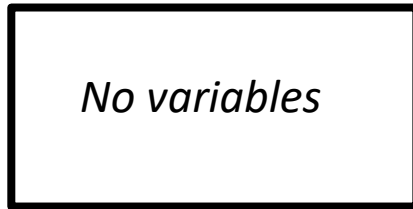
terminal

```
> python maxmax.py
```

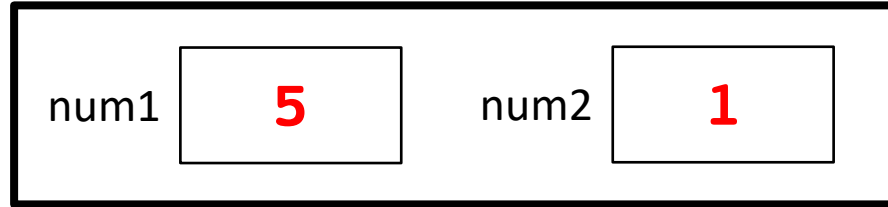


Multiple Return Statements

main memory



max memory

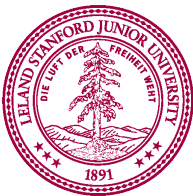


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

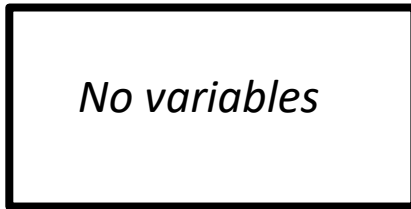
terminal

```
> python maxmax.py
```

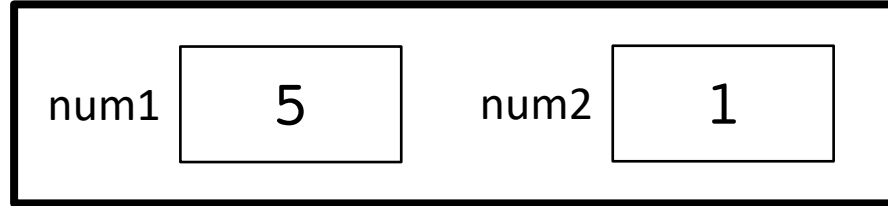


Multiple Return Statements

main memory



max memory

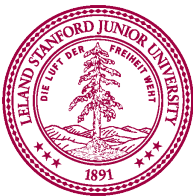


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

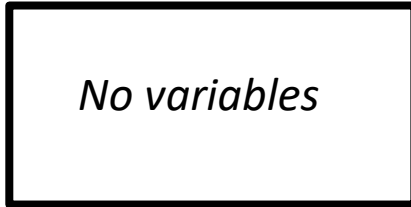
terminal

```
> python maxmax.py
```

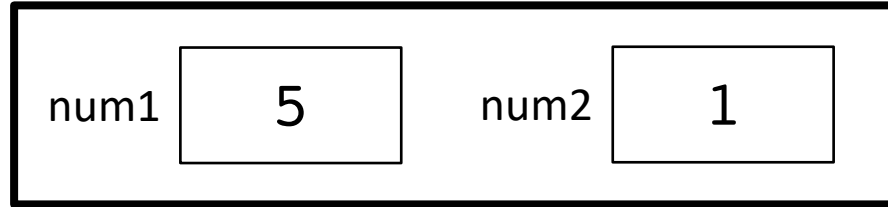


Multiple Return Statements

main memory



max memory

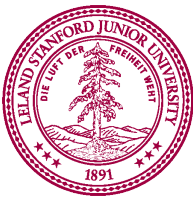


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

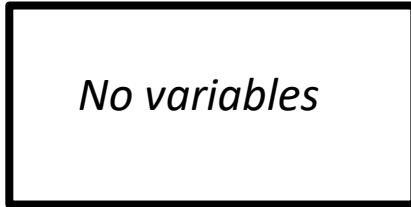
terminal

```
> python maxmax.py
```

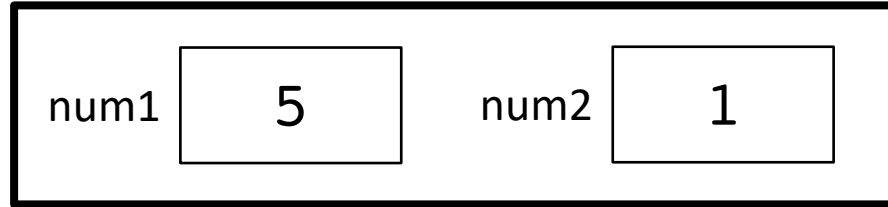


Multiple Return Statements

main memory



max memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

5

```
def main():  
    larger = max(5, 1)
```

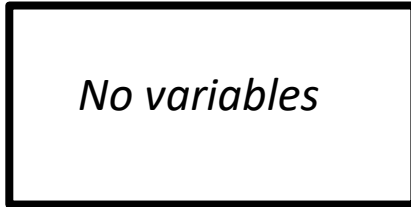
terminal

```
> python maxmax.py
```

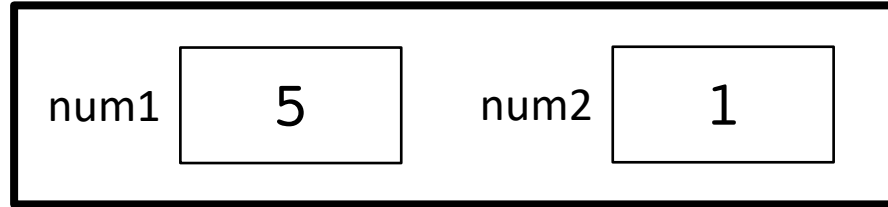


Multiple Return Statements

main memory



max memory

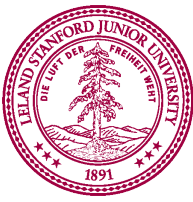


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

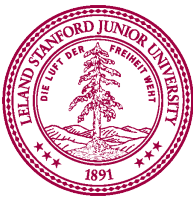
No variables

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

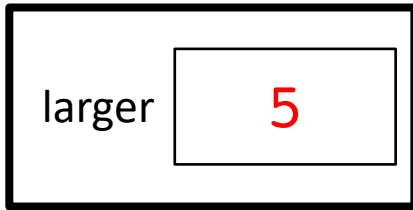
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory

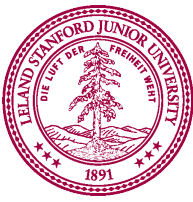


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```

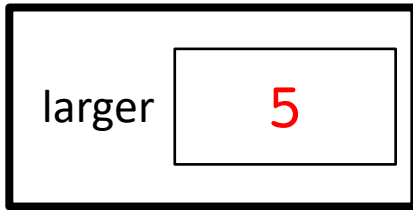
terminal

```
> python maxmax.py
```



Multiple Return Statements

main memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(5, 1)
```



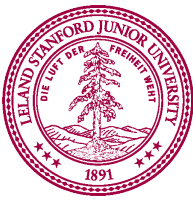
terminal

```
> python maxmax.py
```



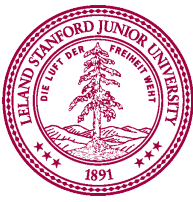
Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(5, 1)
```



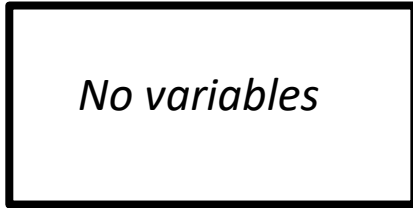
Multiple Return Statements

```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2  
  
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

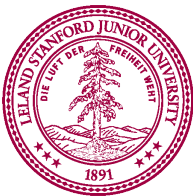
main memory



No variables

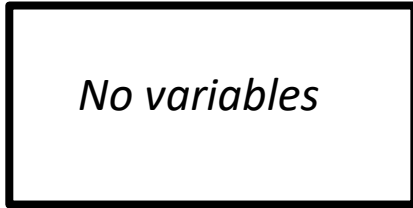
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

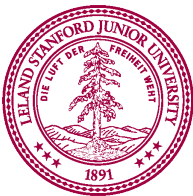
main memory



No variables

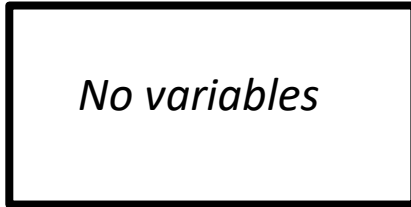
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

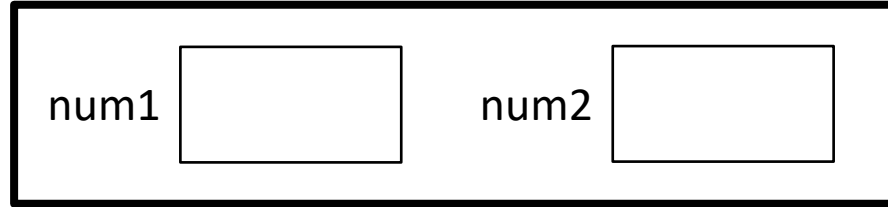


Multiple Return Statements

main memory

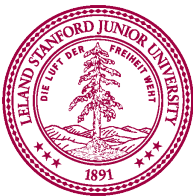


max memory



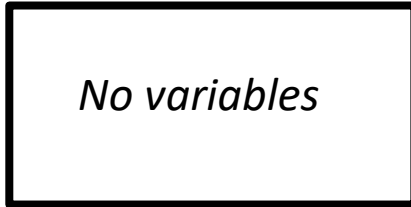
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

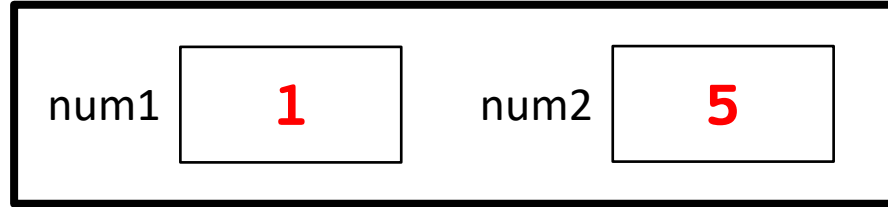


Multiple Return Statements

main memory

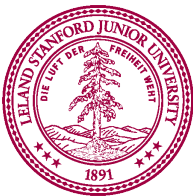


max memory



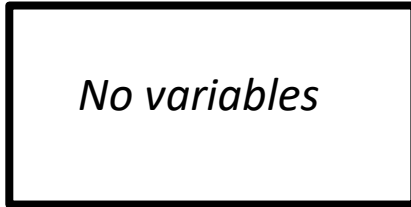
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

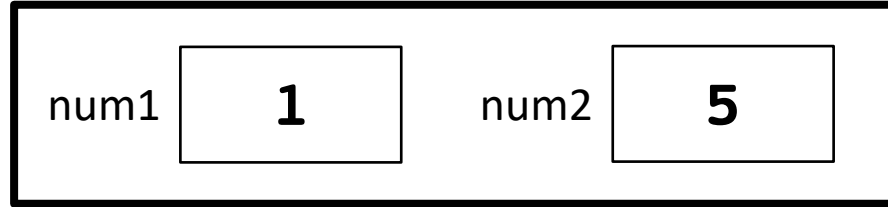


Multiple Return Statements

main memory

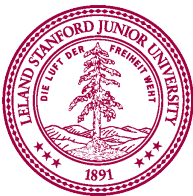


max memory



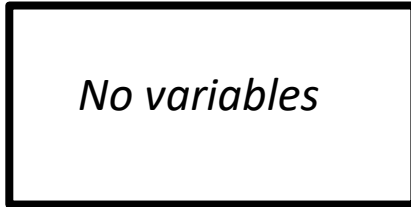
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```

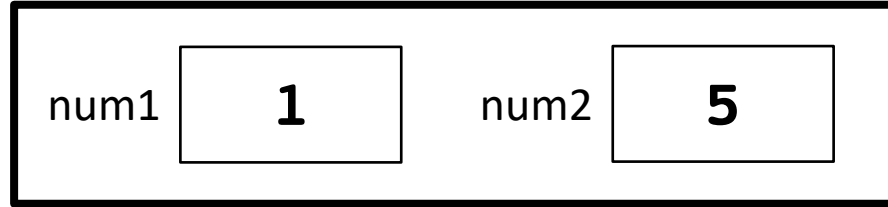


Multiple Return Statements

main memory



max memory

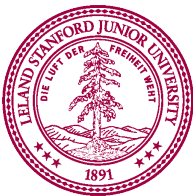


```
def max(num1, num2):  
    if num1 >= num2:  
        return num1
```

```
    return num2
```

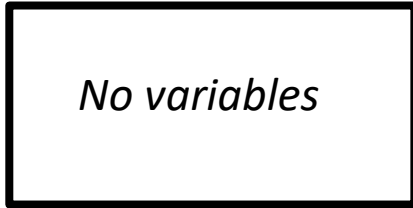
5

```
def main():  
    larger = max(1, 5)
```



Multiple Return Statements

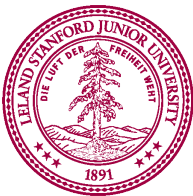
main memory



No variables

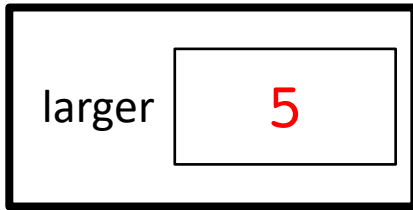
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



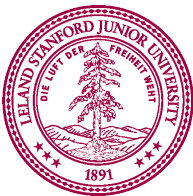
Multiple Return Statements

main memory



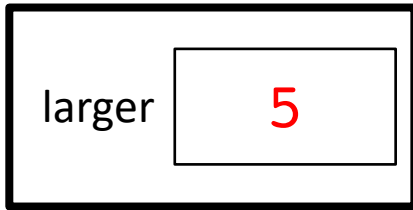
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



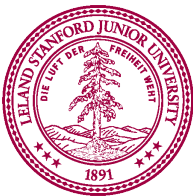
Multiple Return Statements

main memory



```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
  
    return num2
```

```
def main():  
    larger = max(1, 5)
```



Function for IO

```
Chris@ndoto Lecture7 % python io.py
```

A large, stylized letter 'I' is formed by repeating the text 'io' many times. The characters are arranged in a grid-like pattern, with the 'i' and 'o' of one 'io' pair positioned slightly to the left and above the 'i' and 'o' of the next pair, creating a sense of depth and movement. The overall shape is a tall, narrow vertical column that tapers slightly at the top and bottom, with the 'io' pairs following the contour of the letter.

I give you

```
print_no_return
```

What functions do you define?



A Full Program

```
MAX_NUM = 4
```

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

Understand the Mechanism



```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

0

```
def main():  
    for i in range(MAX NUM):  
        print(i, factorial(i))
```

i

0

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 0

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

0

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n

0

result

i

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n

0

result

1

i

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n

0

result

1

i

1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n

0

result

1

i

1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n

0

result

1

i

1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

1

i

0

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

1

i

0

0 1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 1

0 1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 1

0 1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 1

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n result i

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n result i

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n 1 result 1 i 1

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n 1 result 1 i 1

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n 1 result 1 i 1

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n 1 result 1 i 2

0 1

```
def factorial(n):  
    result = 1  
    for i in range(1, n+1):  
        result *= i  
  
    return result
```

n 1 result 1 i 2

0 1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

1

i

1

0

1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

1

i

1

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 2

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 2

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

2

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

2

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

2

i

2

0	1
1	1

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

2

i

2

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 3

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 3

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 3

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 3

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

6

i

3

0	1
1	1
2	2

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

6

i

3

0	1
1	1
2	2
3	6

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i

4

0	1
1	1
2	2
3	6

```
def main():  
    for i in range(MAX_NUM):  
        print(i, factorial(i))
```

i 4

0	1
1	1
2	2
3	6

Parameters



Every time a
function is called,
new memory is
created for the
call.

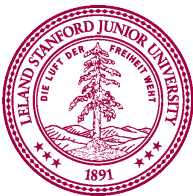
Bad Times With functions

// NOTE: This program is buggy!!

```
def add_five(x):  
    x += 5
```

```
def main():  
    x = 3  
    add_five(x)  
    print("x = " + x)
```

Let's "trace" this
program on the slides

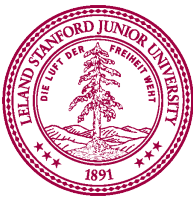


Good Times With functions

// NOTE: This program is **feeling just fine...**

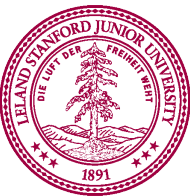
```
def add_five(x):  
    x += 5  
    return x
```

```
def main():  
    x = 3  
    x = add_five(x)  
    print("x = " + x)
```





For primitives:
Variables are **not**
passed when you
use parameters.
Values are passed



Pass by “Value”



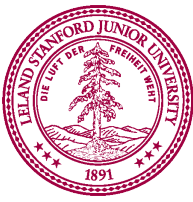
- Thanks Mehran

More Examples

Changed Name

```
def main():  
    num = 5  
    cow(num)
```

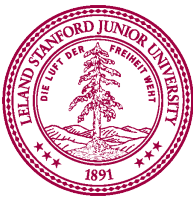
```
def cow(grass):  
    print(grass)
```



Same Variable Name

```
def main():  
    num = 5  
    cow()  
    print(num)
```

```
def cow():  
    num = 10  
    print(num)
```

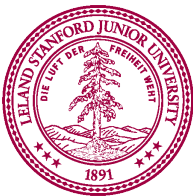


No functions in functions

```
def main():  
    print("hello world")  
    def say_goodbye():  
        print("goodbye!")
```



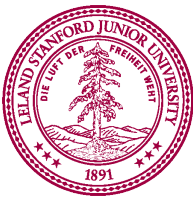
Technically legal, but often a sign at the start that you are confusing definition and calling



No functions in functions

```
def main():  
    print("hello world")  
    say_goodbye()
```

```
def say_goodbye():  
    print("goodbye!")
```



Learn How To:

1. Write a function that takes in input
2. Write a function that gives back output
3. Trace function calls using stacks

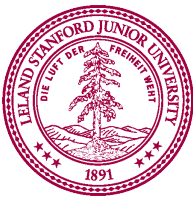


Remember Booleans?

Boolean Variable

```
karelIsAwesome = true
```

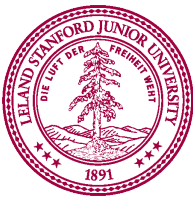
```
myBool = 1 < 2
```





Is Square

```
def main():  
    for i in range(100):  
        if is_square(i):  
            print(i)
```

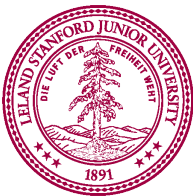


Boolean Return

```
def main():  
    for i in range(100):  
        if is_square(i):  
            print(i)
```



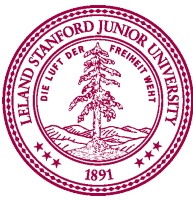
```
def is_square(x):  
    root = math.sqrt(x)  
    if is_whole(root):  
        return True  
    else:  
        return False
```



Boolean Return

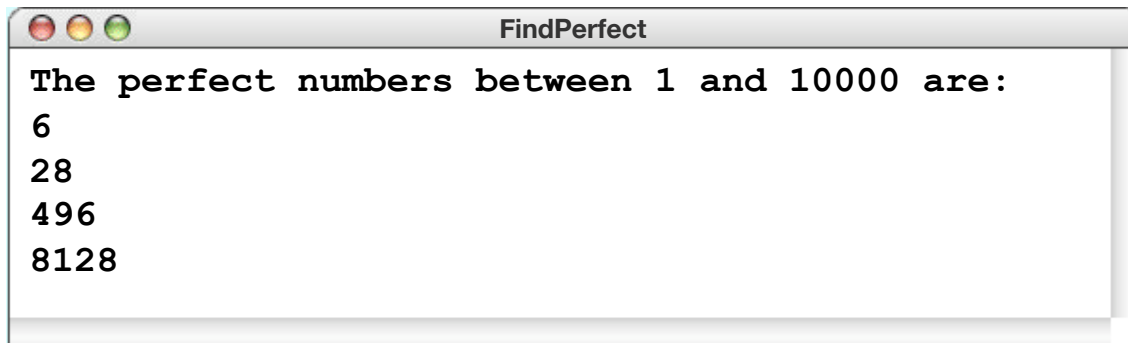
```
def main():  
    for i in range(100):  
        if is_square(i):  
            print(i)
```

```
def is_square(x):  
    root = math.sqrt(x)  
    return is_whole(x)
```



Extra Exercise

- Greek mathematicians took a special interest in numbers that are equal to the sum of their proper divisors (a proper divisor of n is any divisor less than n itself). They called such numbers *perfect numbers*. For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is the sum of 1, 2, 4, 7, and 14.
- Design and implement a Python program that finds all the perfect numbers between two limits. For example, if the limits are 1 and 10000, the output should look like this:



```
FindPerfect  
The perfect numbers between 1 and 10000 are:  
6  
28  
496  
8128
```

